Apprentissage Statistique

Détection de structures communautaires dans des réseaux

Rédigé par

PRALON Nicolas

CÔME Olivier

SENE Assane

Table des matières

Introduction

De multiples réseaux, y compris les réseaux sociaux, les réseaux informatiques, se divisent plus ou moins naturellement en communautés. La détection de cette structure sous-jacente aux réseaux constitue un problème actuel, et de nombreuses approches ont été développées pour y répondre.

Dans ce rapport nous allons présenter une approche communément utilisée en apprentissage non supervisé, permettant de mesurer la validité d'un partitionnement du réseau, les défaillances de cette approche et la mise en pratique des méthodes utilisées pour y répondre.

Concept de Modularité

L'étude d'éventuelles structures communautaires dans des réseaux peut formellement être présentée par l'étude de graphe. Ainsi nous considérons un réseau comme un graphe, et émettons certaines hypothèses à notre étude :

$$Soit\ G = (V, E),\ \text{un graphe tels que}$$

$V = \{v_1, \cdots, v_p\}$ l'ensemble des nœuds
$E \subset \{(v_i, v_j)_{i,j \in \{1,\cdots,p\}} | i \neq j\} = V \times V$ l'ensemble des arêtes du graphe

G est un graphe simple, non orienté, non pondéré, non labélisé.

Avant de décrire l'idée mise en œuvre pour la détection de communautés, donnons quelques définitions.

Définition 1 (Densité)**.** On appel densité d'un graphe la valeur

$$D_G = \frac{|E|}{\frac{p^2-p}{2}}$$

La densité d'un graphe correspond à la fréquence d'arêtes dans le graphe, il rend compte de la connexion entre les nœuds.

Définition 2 (Degré)**.** On appel degré d'un nœud i la valeur

$$d_i = |\{(v_i, v_j) \in E | j \in \{1, \cdots, p\}\}|$$

et correspond au nombre de voisin du noeud i.

Définition 3 (Modèle nul)**.** On appelle modèle nul d'un graph G, le graph G^* dont les $|E| = m$ arêtes ont été distribuées aléatoirement entre les nœuds de G

Le modèle nul joue un modèle de référence pour lequel il n'existe aucune structure communautaire dans le réseau.

Revenons sur la question de détection de communautés et abordons la par étape.
Soit G un graphe.
Par simplicité nous souhaitons déterminer s'il existe une division du graphe G en deux communautés. Une approche intuitive est de chercher deux groupes de nœuds pour lesquels on cherche à maximiser le nombre d'arêtes entre les nœuds du groupe et à minimiser le nombre d'arêtes entre nœuds de groupes différents.

Cependant le simple comptage des arêtes n'est pas un bon moyen de quantifier le concept de structure communautaire. Si l'on choisissait pour groupe le graphe et pour second groupe un ensemble vide, ce partitionnement serait alors optimal, mais ne répond pas au problème.

Le concept de modularité que nous allons présenter est l'idée sous laquelle, si il existait éventuellement une structure dans un graphe, alors en le comparant à son modèle nul pour lequel il n'existe aucune structure sous-jacente, nous devrions raisonnablement observer une différence entre les deux modèles.

Définition 4 (Modularité)**.** Soit $(C_1, \cdots, C_K)$ une partition de V
On définit la modularité Q de la partition comme

$$Q(C_1, \cdots, C_K) = \frac{1}{2m} \sum_{k=1}^{K} \sum_{(v_i, v_j) \in C_k} (\mathbb{1}_{v_i, v_j) \in E} - P_{i,j})$$

avec $P_{i,j} = \frac{d_i d_j}{2m}$ la probabilité que i et j soient connectés sous le modèle nul, et $2m = \sum_{i=1}^{p} d_i$

La partition $(C_1, \cdots, C_K)$ est une bonne partition, selon la valeur de sa modularité, si la densité obervée dans chacun des groupes est plus élevé que la densité attendue de ce groupe, dans le model nul.

Il s'agit alors de maximiser la modularité pour déterminer du meilleur partionnement, selon cette idée.

Une critique peut déjà être approrté, en présence d'un réseau de très grande taille, la quantité

$$\frac{\sum_{(v_i, v_j) \in C_k} (\mathbb{1}_{v_i, v_j) \in E} - P_{i,j})}{2m}$$

représentant une petit communauté, n'est pas significatif et même bien définie, l'optimsation de la modularité risque de ne pas distinguer cette partition.

De plus la maximisation de la modularité est un problème NP-complet, il convient donc d'utiliser des algoritmes détournés afin d'estimer ce maximum.

Méthode de Louvain

Nous présentons dans cette partie la méthode Louvain permettant d'approcher la partition de modularité maximale, il s'agit d'un algorithme glouton avec une complexité temporelle en $\mathcal{O}(nlog(n))$.

L'algorithme de Louvain est une méthode itérative et comporte deux étapes successives

- Choix d'un parcours aléatoire sur chaque noeud du réseau
- Affécter chaque noeud à sa propre communauté
- La communauté du noeud i est agrégée à celle d'un noeud voisin j si, la différence de la modularité de la suppréssion de i de sa communauté et son déplacement à celle de j est positive et plus grande que celle de ses autres voisins, cette dérnière étant la quantité

  $\Delta Q = Q(C_1, \cdots, C_i, \cdots, C_j, \cdots, C_p) - Q(C_1, \cdots, C_{i,j}, \cdots, C_p)$
- On obtient un nouveau réseau dont chaque communauté formée devient un noeud du nouveau réseau et dont les arêtes sont pondérées (somme des arêtes entre les groupes).
- On itère la procedure jusqu'à $\Delta Q = 0$

Cette approche est communément utilisée et permet d'obtenir des résultats en un bon temps de calcul. Cependant cette méthode peut donner des résultats différents (nombre de communautés et nœuds dans les communautés) suivant les chemins choisis au court des étapes.

Expériences numériques avec la librairie *igraph*

Dans la section suivante, nous utilisons la librairie *igraph* afin de réaliser les expériences numériques. Nous allons dans un premier temps nous aider du célèbre graphe "*Zachary*". Ce graphe représente tout simplement les relations sociales entre les 34 (nombre de nœuds du graphe) individus d'un club de karaté. Une arête va représenter une relation sociale entre deux individus. Il y a au total 78 arêtes dans ce graphe. La fonction de la librairie *igraph* qui va nous permettre de détecter les différentes structures communautaires est la suivante : $community\_edge\_betweenness()$.
Comme on peut le voir sur la figure 1, l'algorithme détecte qu'il y a 5 communautés différentes dans le graphe. On remarque que ces différentes communautés sont relativement bien séparées. On les distingue clairement.

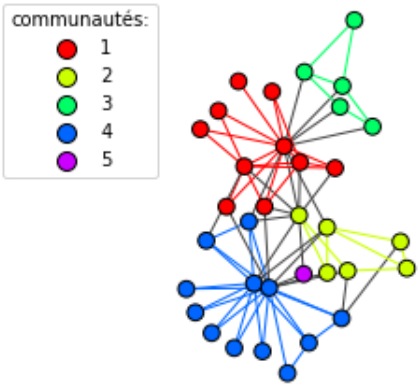

FIGURE 1 – Structures communautaires du graphe du club de karaté Zachary

Dans ce deuxième exemple, nous avons générer un graphe de manière aléatoire via la méthode $Erdos\_Renyi(n, m)$ où n représente le nombre de sommets et m le nombre d'arêtes du graphe. Le modèle de Erdos-Rényi choisit un graphe uniformément au hasard parmi la collection de tous les graphes qui ont n nœuds et M arêtes. Comme on peut le voir sur la figure 2, l'algorithme détecte qu'il y 9 communautés différentes dans le graphe. On remarque que ces différentes communautés sont mal séparées. On ne les distingue pas clairement.

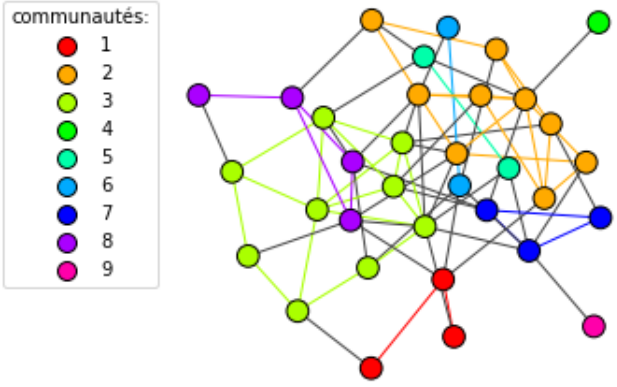

FIGURE 2 – Structures communautaires d'un graphe généré aléatoirement

Annexes

Dans ces annexes vous trouverez une image présentant le fonctionnement de l'algorithme de Louvain ainsi que nos scripts qui ont été implémentés dans le but de faire de la détection de structures communautaires sur les deux graphes énoncés précédemment.

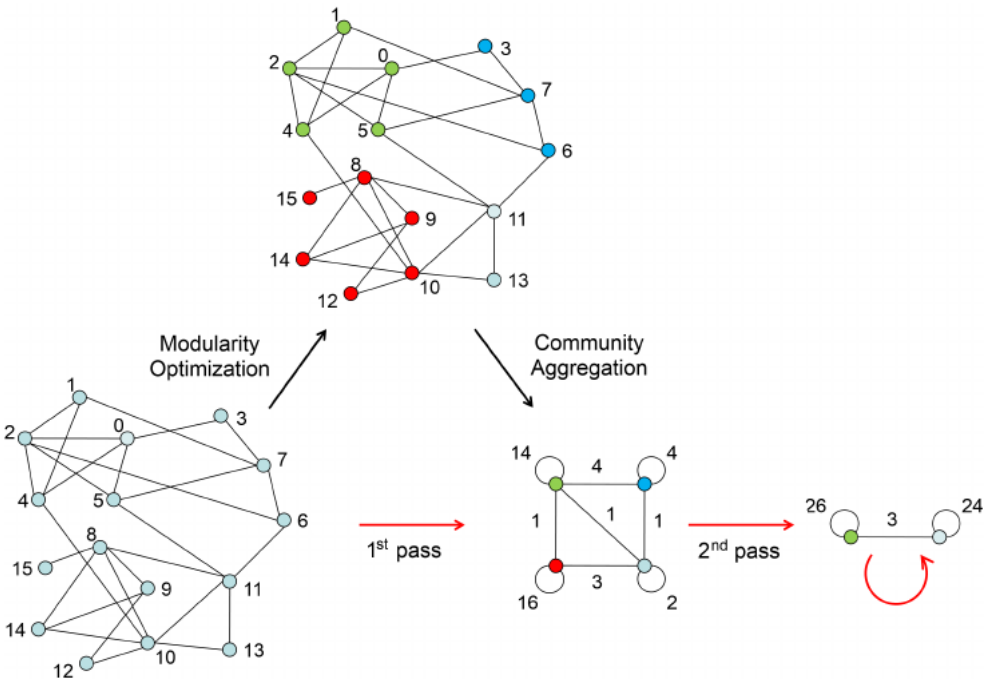

FIGURE 3 – Application de l'algorithme de Louvain

Ci dessous le script utilisé pour faire de la détection de communautés sur le graphe "Zachary"

```
import igraph as ig
import matplotlib.pyplot as plt

# On charge le celebre graphe du club de karate de Zachary
G_karateClub = ig.Graph.Famous("Zachary")

# On utilise la centralite d'intermediarite
# pour la detection de communaute
communities = G_karateClub.community_edge_betweenness()
# On convertit l'objet communities en un objet VertexClustering
# pour la representation graphique
communities = communities.as_clustering()
# On va colorier chaque sommet et arete en fonction de leur appartenance
# a une communaute
numberOfCommunities = len(communities)
ColorPalette = ig.RainbowPalette(n=numberOfCommunities)
for i, c in enumerate(communities):
    G_karateClub.vs[c]["color"] = i
    community_edges = G_karateClub.es.select(_within=c)
    community_edges["color"] = i
    print((i,c))
# On affiche le graphe avec les sommets et les aretes colorees en fonction
# de leur appartenance a une communaute
fig, ax = plt.subplots()
ig.plot(
    communities,
    palette=ColorPalette,
    edge_width=1,
    target=ax,
    vertex_size=0.3,
)

# On cree la legende qui indiquera les couleurs associees a chaque communautes
G_legend = []
for i in range(numberOfCommunities):
    handle = ax.scatter(
        [], [],
        s=100,
        facecolor=ColorPalette.get(i),
        edgecolor="k",
        label=i+1,
    )
    G_legend.append(handle)

ax.legend(
    handles=G_legend,
    title='communautes:',
    bbox_to_anchor=(0, 1.0),
    bbox_transform=ax.transAxes,
)

# On sauvegarde le trace du graphe au format png
#plt.savefig('G_zachary.png')
```

Ci dessous le script utilisé pour faire de la détection de communautés sur le graphe généré aléatoirement avec la méthode de $Erdos\_Renyi(n, m)$.

```
import random as rd
import matplotlib.pyplot as plt
import igraph as ig

rd.seed(123)

myGraphe = ig.Graph.Erdos_Renyi(34, m=78, directed=False, loops=False)

# On utilise la centralite d'intermediarite
# pour la detection de communaute
communitiesOfmyGraphe = myGraphe.community_edge_betweenness()
# On convertit l'objet communities en un objet VertexClustering
# pour la representation graphique
communitiesOfmyGraphe = communitiesOfmyGraphe.as_clustering()
# On va colorier chaque sommet et arete en fonction de leur appartenance
# a une communaute
numberOfCommunities = len(communitiesOfmyGraphe)
ColorPalette = ig.RainbowPalette(n=numberOfCommunities)
for i, c in enumerate(communitiesOfmyGraphe):
    myGraphe.vs[c]["color"] = i
    community_edges = myGraphe.es.select(_within=c)
    community_edges["color"] = i
    print((i,c))
# On affiche le graphe avec les sommets et les aretes colorees en fonction
# de leur appartenance a une communaute
fig, ax = plt.subplots()
ig.plot(
    communitiesOfmyGraphe,
    palette=ColorPalette,
    edge_width=1,
    target=ax,
    vertex_size=0.3,
)

# On cree la legende qui indiquera les couleurs associees a chaque communautes
G_legend = []
for i in range(numberOfCommunities):
    handle = ax.scatter(
        [], [],
        s=100,
        facecolor=ColorPalette.get(i),
        edgecolor="k",
        label=i+1,
    )
    G_legend.append(handle)

ax.legend(
    handles=G_legend,
    title='communautes:',
    bbox_to_anchor=(0, 1.0),
    bbox_transform=ax.transAxes,
)

#plt.savefig('random_network2.png')
```

Bibliographie

[1] WikiStat. An introduction to network inference and mining,*Article*
http://www.nathalievialaneix.eu/doc/pdf/wikistat-network_compiled.pdf

[2] PNAS. Modularity and community structure in networks (2015),*Article*
https://www.pnas.org/doi/10.1073/pnas.0601602103#abstract

[3] Wikipédia (2022). Méthode de Louvain, *Article*
https://fr.wikipedia.org/wiki/MÃľthode_de_Louvain

[4] igraph, *Documentation*
https://igraph.org/python/versions/latest/

[5] igraph, *Documentation*
https://igraph.org/python/versions/latest/tutorials/visualize_communities/visualize_communities.html
[6] igraph, *Tutoriel*
https://igraph.org/python/api/latest/igraph._igraph.GraphBase.html#Erdos_Renyi